COMMANDS — FROM ZERO TO LIVE (PowerShell, Azure + Docker)
Project: FastAPI + LangChain + Azure AI Search (RAG) on Azure Container Apps

How to use this document

• Copy/paste commands into a PowerShell terminal.
• Replace any placeholders in ANGLE_BRACKETS, e.g., <YOUR_OPENAI_KEY>.
• Lines that start with "Why:" explain the purpose in plain language.
• All commands shown are the corrected versions you successfully used (with notes on earlier errors and how we fixed them).

---

0) Local prerequisites (once)

```
python -V
pip -V
pip install -r requirements.txt
```

Why: Confirms Python and installs project dependencies for local runs.

```
uvicorn app.api:app --reload --port 8000
```

Why: Starts the API locally for smoke tests (Swagger at http://127.0.0.1:8000/docs).

```
python -m app.ingest.load_docs
```

Why: Creates the Azure AI Search index (if missing) and uploads your document chunks.

---

1. Log in to Azure and pick the right subscription

   ```
   az login
   ```

   Why: Interactive sign-in. If your tenant enforces MFA or you have multiple tenants, use the tenant flag:

```
az login --tenant 1794c864-a6bb-4f2d-8c5b-9f80e261ba6b
```

Why: Ensures you authenticate against the correct Azure AD tenant (your "Default Directory"). This fixed the early "MFA/tenant" login issues.

```
az account list -o table
```

Why: Lists all subscriptions you can access and shows their IDs.

```
az account set --subscription f12f3b4e-979e-46ed-855a-d909c80ec0ec
```

Why: Explicitly sets the current subscription to "Azure subscription 1". This fixed the "SubscriptionNotFound" errors when creating resources.

```
az account show -o table
```

Why: Verifies you're on the expected subscription (name/tenant).

Common login errors you fixed
• AADSTS50076 MFA required → solved by logging in with the tenant flag and completing MFA.
• "Subscription not found" → solved by setting the subscription explicitly as above.

---

2) Create a resource group (logical container)

```
$rg = "rg-rag"
$loc = "ukwest"
az group create --name $rg --location $loc
```

Why: A resource group keeps all Azure resources for this project in one place (easy to manage and delete).

---

3) Register required resource providers (one-time per subscription)

```
az provider register -n Microsoft.Search --wait
az provider register -n Microsoft.ContainerRegistry --wait
az provider register -n Microsoft.App --wait
az provider register -n Microsoft.OperationalInsights --wait
```

Why: Allows your subscription to create Azure AI Search, Azure Container Registry (ACR), Azure Container Apps, and Log Analytics resources. This fixed the earlier "subscription is not registered to use namespace …" errors.

Check status (optional):

```
az provider show -n Microsoft.Search --query registrationState -o tsv
az provider show -n Microsoft.ContainerRegistry --query registrationState -o tsv
az provider show -n Microsoft.App --query registrationState -o tsv
az provider show -n Microsoft.OperationalInsights --query registrationState -o tsv
```

---

4) Create Azure AI Search (vector index backend)

```
$searchName = "ragsearch709278" # You already created this successfully
az search service create --name $searchName
--resource-group $rg --sku basic
--location $loc
```

Why: Creates the managed search service you'll use as a vector store.

Get the admin key (needed by your app and ingestion script):

```
$searchKey = az search admin-key show -g $rg --service-name $searchName --query primaryKey -o tsv
$searchEndpoint = "https://$searchName.search.windows.net"
```

Why: You'll pass $searchEndpoint and $searchKey to the app as secrets.

Quick sanity check (optional; 403 is expected without a key):

```
curl https://$searchName.search.windows.net
```

Why: Confirms DNS/ingress; the 403 proves the endpoint exists but denies anonymous requests.

---

5) (Optional) Create a Storage Account
$stg = "ragstg709278"
az storage account create `--name $stg`
--resource-group $rg `--location $loc`
--sku Standard_LRS
Why: Blob/Table/Queue storage for future ingestion pipelines or raw document storage. You registered Microsoft.Storage earlier when needed.

---

6) Create an Azure Container Registry (ACR)
$acr = "acrrag709278"
az acr create -g $rg -n $acr --sku Basic
Why: Private Docker registry to hold your API image.

(If you use admin creds for one-off pulls/pushes)
az acr update -n $acr --admin-enabled true
$acrUser = az acr credential show -n $acr --query username -o tsv
$acrPass = az acr credential show -n $acr --query passwords[0].value -o tsv
Why: Enables username/password on ACR for quick integration. In production, prefer managed identities/roles.

---

7) Build and push the container image to ACR
Option A — Build in ACR (simple, avoids local Docker login):
$imageName = "rag-azure"
$tag = "v1"
az acr build -r $acr -t "rag-azure:v1" .
Why: Builds your Dockerfile server-side and stores the image in ACR as acrrag709278.azurecr.io/rag-azure:v1.

(Notes you fixed: In PowerShell, variables with a colon inside a string like $imageName:$tag can confuse the parser. You resolved it by quoting the entire tag value “rag-azure:v1” instead of using variables around the colon.)

Option B — Build locally (if you prefer):
docker build -t rag-azure:v1 .
docker tag rag-azure:v1 "$acr.azurecr.io/rag-azure:v1"
az acr login -n $acr
docker push "$acr.azurecr.io/rag-azure:v1"
Why: Standard local build/push flow. Either option is fine; you used Option A.

---

8) Create a Container Apps environment
$envName = "aca-env-rag"
az containerapp env create -g $rg -n $envName -l $loc
Why: Provision the runtime environment for Container Apps. (Earlier error "Subscription not registered for Microsoft.OperationalInsights" was fixed by registering that provider.)

(Optional auto-install for extensions)
az config set extension.use_dynamic_install=yes_without_prompt
Why: Lets az auto-install missing CLI extensions like containerapp.

---

9) Create the Container App (public URL)
$app = "aca-rag"
$imageRef = "$($acr).azurecr.io/rag-azure:v1"

```
az containerapp create -g $rg -n $app
 --environment $envName --image $imageRef
 --ingress external --target-port 8080 --min-replicas 0 --max-replicas 1
 --registry-server "$($acr).azurecr.io" --registry-username $acrUser
 --registry-password $acrPass
```

Why: Deploys your container, exposes it publicly on port 8080, and sets scale-to-zero when idle. The registry credentials let Container Apps pull from ACR. (Earlier you saw "expected one argument" because $acrUser/$acrPass were empty; enabling ACR admin and retrieving credentials fixed that.)

Get the public URL (FQDN):
$FQDN = az containerapp show -g $rg -n $app --query properties.configuration.ingress.fqdn -o tsv
https://aca-rag.wittysand-46da5683.ukwest.azurecontainerapps.io/
Why: Copy this URL, append /docs in a browser to see the API Swagger UI online.

---

10) Add runtime secrets (lowercase names) and map to env vars
Important: Container Apps secret names must be lowercase, alphanumeric or "-".

Set secrets:
$az_search_endpoint = $searchEndpoint
$az_search_api_key = $searchKey
$openai_api_key = "<YOUR_OPENAI_KEY>"
$jwt_secret = "change-me"
$jwt_alg = "HS256"
$az_search_index = "docs-index"

az containerapp secret set -g $rg -n $app `--secrets`
az-search-endpoint="$az_search_endpoint"
`az-search-api-key="$az_search_api_key"`
openai-api-key="$openai_api_key" `jwt-secret="$jwt_secret"`
jwt-alg="$jwt_alg" `
az-search-index="$az_search_index"
Why: Stores sensitive values as Container Apps secrets. Earlier error “invalid secret name” occurred because names were uppercase with underscores; using lowercase and hyphens fixed it.

Map secrets to environment variables your app reads:
az containerapp update -g $rg -n $app `--set-env-vars`
AZ_SEARCH_ENDPOINT=secretref:az-search-endpoint
`AZ_SEARCH_API_KEY=secretref:az-search-api-key`
OPENAI_API_KEY=secretref:openai-api-key `JWT_SECRET=secretref:jwt-secret`
JWT_ALG=secretref:jwt-alg `
AZ_SEARCH_INDEX=secretref:az-search-index
Why: Injects the secret values into your container as environment variables without exposing the raw secrets.

Restart (usually automatic after update; to force):
az containerapp restart -g $rg -n $app

Test live:
"https://aca-rag.wittysand-46da5683.ukwest.azurecontainerapps.io/docs"
Why: Open Swagger, POST /token to get a bearer token, then POST /query with your question.

---

11) Day-2 operations (useful commands)

Logs and health
az containerapp logs show -g $rg -n $app --follow
Why: Streams container logs for troubleshooting.

Invoke the health endpoint (quick check):
Invoke-WebRequest -UseBasicParsing
"https://aca-rag.wittysand-46da5683.ukwest.azurecontainerapps.io/health"
Why: Returns { "status":"ok" } if the app is up.

Scaling (manual)
az containerapp update -g $rg -n $app --min-replicas 1 --max-replicas 2
Why: Keeps at least one replica warm; allows bursts up to 2.

Deploy a new image version

Build new image version in ACR:

```
az acr build -r $acr -t "rag-azure:v2" .
```

Point the app at the new image:

```
az containerapp update -g $rg -n $app --image "$($acr).azurecr.io/rag-azure:v2"
```
Why: Zero-downtime revision rollout. You can roll back to prior revisions in Container Apps if needed.

Update secrets (e.g., rotate OpenAI key)
```
az containerapp secret set -g $rg -n $app --secrets openai-api-key="<NEW_KEY>"
az containerapp update -g $rg -n $app --set-env-vars
OPENAI_API_KEY=secretref:openai-api-key
```
Why: Rotates secrets without rebuilding the image.

Show app details (URL, image, secrets names)
```
az containerapp show -g $rg -n $app -o json
```
Why: Inspect configuration.

---

12) Local developer workflow (quick reference)

Start API locally:
```
uvicorn app.api:app --reload --port 8000
```

Open Swagger:
http://127.0.0.1:8000/docs

Get token then query:
- POST /token → copy access_token
- Authorize → paste bearer token
- POST /query?q="your question"

Load documents to Search:
```
python -m app.ingest.load_docs
```

Delete documents by filename:
```
python -m app.ingest.delete_docs --filename about.txt
```

Verify OpenAI key (sanity, optional):
```
python tests/api_key_test.py
```

---

13) Troubleshooting you already solved

"Subscription not found" when creating resources
Fix: az account set --subscription <SUBSCRIPTION_ID> to ensure you're on the correct subscription.

"MFA / tenant error" on login
Fix: az login --tenant <TENANT_ID> and complete MFA.

"Subscription is not registered to use namespace …"
Fix: az provider register -n --wait for the required providers (Search, ContainerRegistry, App, OperationalInsights).

PowerShell parsing error with image tags (colon in variable expansion)
Fix: Quote the entire tag string in PowerShell, e.g., -t "rag-azure:v1" rather than using $imageName:$tag inline.

Container Apps secret name invalid
Fix: Use lowercase and hyphens (e.g., az-search-endpoint), then map to env vars via secretref.

Cannot pull from ACR (missing credentials)
Fix: Enable admin on ACR and fetch credentials (or use managed identity/role assignments).
az acr update -n $acr --admin-enabled true
az acr credential show -n $acr …

Container Apps environment creation blocked by Log Analytics provider
Fix: Register Microsoft.OperationalInsights and re-run env creation.

Search endpoint returns 403
Explanation: Expected without API key; confirms endpoint exists. Use the admin key from az search admin-key show for programmatic access.

OpenAI 401/429 during ingestion
Explanation: 401 = invalid/expired key; 429 = insufficient quota. You verified the key with tests/api_key_test.py and managed usage on the OpenAI dashboard.

---

14) Cleanup (stop charges)
az group delete -n $rg --yes --no-wait
Why: Deletes the entire resource group and everything inside (Search, ACR, Container Apps, etc.). Use only when you're done.

Appendix — What each service does (one-liners)

Resource group (rg-rag): Logical folder for all resources.
 Azure AI Search (ragsearch…): Vector + keyword search engine that stores your chunks and embeddings.
 Azure Container Registry (acrrag…): Private Docker registry hosting your built API images.
 Azure Container Apps (aca-env-rag / aca-rag): Serverless container runtime that exposes your API at a public URL.
 Log Analytics workspace: Collects logs and metrics from the running container.
 Storage account (optional): General-purpose storage for future ingestion and data pipelines.

This is the complete, recruiter-ready command trail from login to live URL, including the exact fixes for issues you hit and why each step matters.